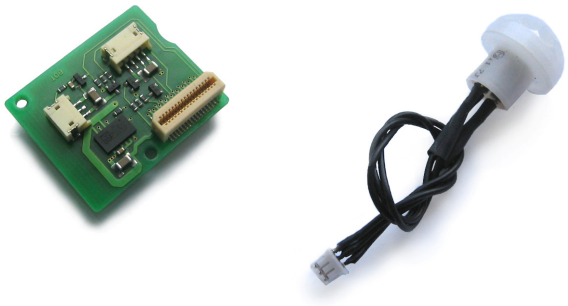


iSense Security Module User Guide

coalesenses
research to innovate

Coalesenses GmbH Maria-Goeppert-Str. 1 23562 Lübeck Germany www.coalesenses.com sales@coalesenses.com

Document history

Version	Date	Changes
1.0	18.05.2009	Initial version
1.1	31.10.2009	Corrected paths for the binary files, added section on the application factory,

Contents

1. About this User Guide 4
2. General Description 5
2.1. Handling and Security Instructions 6
2.2. Accelerometer 6
2.3. Passive infrared (PIR) Sensor 7
3. API Description 8
3.1. SensorHandler API Description 8
3.1.1. handle_sensor 9
3.2. PirSensor API Description 9
3.2.1. Constructor 9
3.2.2. enable 10
3.2.3. disable 10
3.2.4. set_pir_sensor_int_interval 10
3.2.5. set_sensor_handler 11
3.3. BufferDataHandler API Description 11
3.3.1. handle_buffer_data 12
3.4. LisAccelerometer API Description 12
3.4.1. Constructor 13
3.4.2. enable 13
3.4.3. disable 14
3.4.4. set_mode 14
3.4.5. set_treshold 15
3.4.6. set_handler_threshold 15
3.4.7. set_divider 16
3.4.8. set_extended_range 16
3.4.9. set_axes 16
4. Security Module Demo Application 18
4.1. Obtaining the Security Module Demo Application 18
4.2. Compiling the Security Module Demo Application 18
4.2.1. Using Eclipse 18
4.2.2. Using the Command Line 21
4.3. Flashing the Security Module Demo Application 22
4.4. Security Module Demo Application functionality 23
5. References 31

1. About this User Guide

In this user guide,

- files and folders are represented in the Arial typeface,
- code fragments, function names etc. are represented in the `Courier New` typeface,
- GUI elements such as button descriptions etc. are represented in “quotation marks”,
- titles of other documents are presented in *Italic* type.

This manual assumes that the reader has successfully installed the iSense development environment, and obtained the iSense standard firmware. For further information on these steps, consult the *Development Environment Setup User Guide* [1].

In addition, it is assumed that the user is familiar with the use of iShell. For further information on iShell, consult the *iShell User Guide* [2].

For further information on iSense firmware programming concepts and on application development, it is recommended to read the *Writing iSense Applications User Guide* [3].

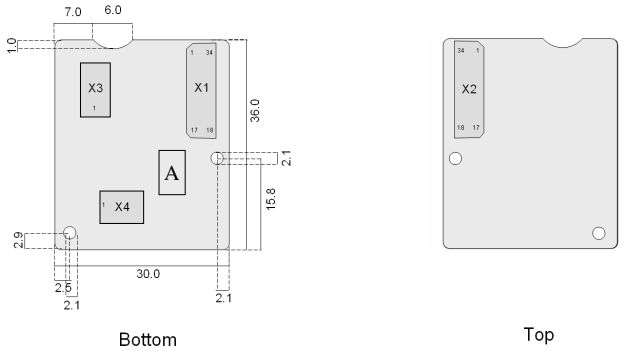
2. General Description

The iSense Security Sensor Module series features a 3-axis accelerometer. A passive infrared (PIR) sensor can be connected via connector X4, a camera module can be attached to connector X3.

The PIR sensor can be used to detect moving objects that feature a temperature different from the environment (such as humans) in distances of up to 10 meters. The sensor offers a wide range of 110° for comprehensive monitoring. If switched on by driving low the DIO9/TIM0_CAP/PIR_ON pin of X1/X2, the PIR sensor generates interrupts on the DIO8/TM0GT/S1_INT pin of X1/X2 upon object detection

The 3-axes accelerometer can be configured to cover accelerations of ±2g or ±6g. In addition to delivering acceleration values via a digital interface, it can generate interrupts on movement or free fall.

In addition, a camera module that can take color pictures with VGA resolution can be attached via connector X3. The images are preprocessed, so they can be scaled down to lower resolutions and compressed according to the JPEG standard. The camera can be activated by driving low the DIO10/TIM0_OUT/CAM_ON pin of X1/X2, and is operated via UART1 by using the DIO20/RXD1 and DIO19/TXD1 pins of X1/X2.



The above figure shows the bottom and top side of the iSense Security Sensor Module, the below table lists its features.

Symbol	*Description*
X1/X2	System connector for plugging an iSense Core Module or other iSense modules
X3	Serial camera connector
X4	Passive infrared sensor connector
A	3-axes ±2g or ±6g accelerometer

2.1. Handling and Security Instructions

Note that all iSense Modules are sensitive to electro-static discharge. Appropriate protection measures have to be taken.

If not protected by an appropriate housing, all iSense components must be protected from humidity, mechanical impact, and short circuiting by contact with conductive materials.

Note that all cable plugs and headers are designed to be extremely compact, and are not intended for frequent plugging and unplugging. Never pull the cables to remove the plugs from the corresponding modules, always pull the plug itself. If required, use an appropriate tool (or the pulling cord if available). Even though all plugs/headers are coded, be sure to pay attention to the proper orientation. Never apply force. Never use cables if their insulation is damaged.

Note that iSense modules are not intended for hot-plugging, (dis-)connecting modules from or to other modules in operation can result in a reset of the Core Module and other undesired effects.

All delivered components are intended for use in research application. For using these components within other application domains, consult coalesenses prior to use.

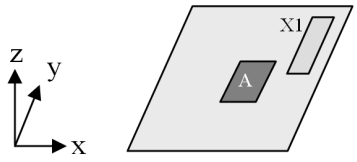
Coalesenses products are not intended for use in life support systems, appliances or systems where malfunction of these products can reasonably be expected to result in personal injury, death or severe property or environmental damage. Coalesenses customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify coalesenses for any damages resulting from such use.

Do not connect components or devices to any iSense component that are not manufactured or distributed by coalesenses without prior consultation of coalesenses. Note that doing so will void the warranty and the declaration of CE conformity regardless of consultation. Coalesenses customers connecting third party devices do so at their own risk and agree to fully indemnify coalesenses for any damages or injury resulting from such use.

2.2. Accelerometer

The 3-axes accelerometer can be configured to cover accelerations of ±2g or ±6g via API calls (see Section 3.4.8).

The accelerometer supports three modes of operation that can also be set via API calls (see Section 3.4.4). In addition to delivering a continuous stream of acceleration values via a digital interface (continuous mode), it can generate interrupts on movement (threshold mode) or free fall (free fall detection mode). For the first two modes, it can be configured which axes (x-axis, y-axis and/or z-axis) are considered (see Section 3.4.9).



In the continuous mode, the accelerometer delivers a stream of accelerometer readings at a frequency of up to 40Hz for all three axes. The sensor will generate interrupts, and the controller has to read out every single data set separately, which it needs to wake up for in case it is sleeping. The data rate can be reduced by configuring a so called frequency divider (a divider of 2 results in a frequency of 20 Hz,

i.e. only every second value is read, a divider of 4 in a frequency of 10 Hz etc.), for details refer to Section 3.4.7. Even though the values are not read, the controller will wakeup nevertheless.

If the sensor node lies flat e.g. on a table, the typical result will be that the x- and y-axis deliver accelerations close to zero, while the z-axis will deliver sensor readings of approximately 1g (due to the earth's gravitation).

In the threshold mode, the sensor generates an interrupt to the controller on the connected Core Module if the sensor values change by more than a configurable threshold on one of the active axes (see Section 3.4.5). In case the controller was sleeping, it is woken by the generated interrupt. After the threshold was exceeded on at least one of the axes, the sensor automatically changes into the continuous mode and delivers a stream of sensor readings until it is set back into the threshold mode.

In the free fall mode, the sensor notifies (i.e. if applicable wakes) the controller by an interrupt if the sensor detects it is in free fall, i.e. all three axes experience an absolute acceleration below the configured threshold. The sensor automatically changes to the continuous mode upon freefall detection.

2.3. Passive infrared (PIR) Sensor

The PIR sensor can be used to detect moving objects that feature a temperature different from the environment (such as humans) in distances of up to 10 meters. The sensor offers a wide range of 110° for comprehensive monitoring.

The PIR sensor can be switched on and off via software API calls (see Section 3.2.2 and 3.2.3). After switching on the sensor, it needs a stabilization time of up to 4 seconds on average. During this time, the interrupt input from the sensor is disabled. Hence, the sensor is dysfunctional during the first 4 seconds after it is switched on.

If switched on by driving low the DIO9/TIM0_CAP/PIR_ON pin of X1/X2, the PIR sensor generates interrupts on the DIO8/TM0GT/S1_INT pin of X1/X2 upon object detection.

In the common case, even single object detections induce a sequence of interrupts on the DIO8/TM0GT/S1_INT pin of X1/X2. To avoid multiple event notifications upon detection of a single detection, the sensor interrupt is disabled for a software configurable time, the so called interrupt suppression interval (see Section 3.2.4) after the first interrupt edge, and the set sensor event handler is notified. After the interrupt suppression interval expired, the interrupt is re-enabled.

3. API Description

The software functionality for operating all features of the Security Module is split into two classes that make use of two additional interface classes.

The `PirSensor` class in the `isense` namespace contains all software functionality for operating the PIR sensor, including

- switching it on and off,
- setting the silence interval, and
- setting a sensor event handler.

The `SensorHandler` interface class in the `isense` namespace defines an abstract pure virtual method that must be implemented by classes that want to be notified about PIR sensor events. It is defined in src/isense/modules/security_module/pir_sensor.h in the iSense directory.

The `LisAccelerometer` class in the `isense` namespace contains all software functionality for operating the accelerometer sensor, including

- switching it on and off,
- configuring the active axes, the sensor operation mode, the threshold interval, the sensor range, and the frequency divider,
- setting a data handler and the data buffer fill state threshold for calling the handler.

The `BufferDataHandler` interface class in the `isense` namespace defines an abstract pure virtual method that must be implemented by classes that want to receive accelerometer data.

The iSense firmware is structured into a large number of software modules that can be (de-)activated separately. The below API description for each function specifies the required modules that must be activated to use the functions, and names both the web compilation module name as well as the according code define. An asterisk (*) indicates that the respective module is activated in the iSense standard firmware.

The API description also indicates the make targets for which the different functions are available.

3.1. SensorHandler API Description

The `isense::SensorHandler` class must be inherited by classes that are to be set at the `PirSensor` class as a `SensorHandler` using the method `PirSensor::set_sensor_handler( SensorHandler* sh)`.

It is defined in src/isense/modules/security_module/pir_sensor.h in the iSense directory.

```
class SensorHandler :
      public iSenseObject
{
      public:
            virtual void handle_sensor() = 0;
};
```

3.1.1. handle_sensor

```
void SensorHandler::handle_sensor();
```

Description:

Overwrite this method if your class inheriting from `SensorHandler` should be called upon PIR sensor events. If your class was set as the PIR sensor event handler before (see Section 3.2.5), it will be called when the sensor detects temperature fluctuations.

Parameters:

none

Required modules:

none

Available for the targets:

all

3.2. PirSensor API Description

The `PirSensor` class in the `isense` namespace contains all software functionality for operating the PIR sensor, including

- switching it on and off,
- setting the interrupt suppression interval, and
- setting a sensor event handler.

The `PirSensor` class is defined in src/isense/modules/security_module/pir_sensor.h in the iSense directory.

3.2.1. Constructor

```
PirSensor::PirSensor(Os &os);
```

Description:

Initializes the `PirSensor` class. By default, the interrupt suppression interval is set to 3.5 seconds. The sensor is not switched on until `enable` is called.

Parameters:

`os` Reference to the operating system class `Os`

Required modules:

Pir Sensor* `#define ISENSE_ENABLE_PIR_SENSOR`

Available for the targets:

JN5139R, JN5139R1

3.2.2. enable

```
bool PirSensor::enable();
```

Description:

Switches on the PIR sensor (regardless of prior state) if a sensor event handler was set before. To allow the sensor to stabilize after power on, interrupts will not be enabled before an interval of 4 seconds passed. If no handler was set, the sensor is not enabled.

Parameters:

none

Returns:

`true` if the sensor was enabled, `false` otherwise.

Required modules:

Pir Sensor* `#define ISENSE_ENABLE_PIR_SENSOR`

Available for the targets:

JN5139R, JN5139R1

3.2.3. disable

```
void PirSensor::disable();
```

Description:

Switches off the PIR sensor (regardless of prior state), and disables the interrupts for the controller IO pin the sensor is connected to.

Parameters:

none

Required modules:

Pir Sensor* `#define ISENSE_ENABLE_PIR_SENSOR`

Available for the targets:

JN5139R, JN5139R1

3.2.4. set_pir_sensor_int_interval

```
void PirSensor::set_pir_sensor_int_interval(uint16 interval );
```

Description:

Sets the PIR sensor interrupt suppression interval (see Section 2.3).

Parameters:

`interval` interrupt suppression interval in milliseconds

Required modules:

Pir Sensor* `#define ISENSE_ENABLE_PIR_SENSOR`

Available for the targets:

JN5139R, JN5139R1

3.2.5. set_sensor_handler

```
void PirSensor::set_sensor_handler( SensorHandler* sh);
```

Description:

Sets the object that `sh` points to as PIR event handler for the `PirSensor`. Note that the class of `sh` must inherit from `isense::SensorHandler` (see Section 3.1) and implement the `handle_sensor()` method. If a different `SensorHandler` was set before, it will be unset, and the new handler will be set.

After the PIR sensor is enabled and an event is detected, `sh->handle_sensor()` will be called.

For unsetting a `SensorHandler` without setting a new handler, simply pass `NULL`. The sensor is disabled upon unsetting the handler.

Parameters:

`sh`	Pointer to the `SensorHandler` that should be called upon event detection. To unset the currently set handler, pass `NULL`.

Required modules:

Pir Sensor* `#define ISENSE_ENABLE_PIR_SENSOR`

Available for the targets:

JN5139R, JN5139R1

3.3. BufferDataHandler API Description

The `isense::BufferDataHandler` class must be inherited by classes that are to be set at the `LisAccelerometer` class as a `BufferDataHandler` using the method `PirSensorLisAccelerometer::set_handler( BufferDataHandler* bdh)`.

It is defined in src/isense/data_handlers.h in the iSense directory.

```
class BufferDataHandler :
      public iSenseObject
{
      public:
            virtual void handle_buffer_data( BufferData* buf_data ) = 0;

};
```

3.3.1. handle_buffer_data

```
void BufferDataHandler::handle_buffer_data( BufferData* data );
```

Description:

Overwrite this method if your class inheriting from `BufferDataHandler` should be called when accelerometer data is available.

Parameters:

`data` Pointer to a `BufferData` structure containing the most recent sensor readings

```
struct BufferData {
      uint32 sec;
      uint16 ms;
      uint8 dim;
      uint8 count;
      int16 buf[BUFFER_DATA_BUFFER_SIZE];
};
```

`sec`, `ms` Time when the first data set of this buffer was captured.

`dim` Number of active axes of each sensor data set (e.g. 2 if the x-axis and the z-axis are activated).

`count` Number of data sets in the buffer.

`buf` Sensor readings. The readings of one data set are following each other directloy, the axes are cascaded in the buffer. If e.g. the x-axis and the z-axis of the sensor are active, `buf[0]` contains the x-value of the first data set, `buf[1]` the z-value of the first data set, `buf[2]` the x-value of the second data set etc.

Required modules:

none

Available for the targets:

all

3.4. LisAccelerometer API Description

The `LisAccelerometer` class in the `isense` namespace contains all software functionality for operating the accelerometer, including

- switching it on and off,
- configuring the active axes, the sensor operation mode, the threshold interval, the sensor range, and the frequency divider,
- setting a data handler and the data buffer fill state threshold for calling the handler.

The `LisAccelerometer` class collects its readings in two buffers. After the first buffer is filled, it is passed to the set data handler, while the sensor meanwhile fills the second buffer. Note that the sensor assumes that the data handler processes the data in the first buffer while the sensor fills the second one, and vice versa. Hence, after the second buffer is passed to the handler, the sensor again starts to fill the first buffer. This in particular means that the handler must not read any but the most recently passed buffer.

The interval in milliseconds between two calls of the handler in the continuous mode can be calculated as follows:

$$\text{interval} = \frac{\text{min[handler\_threshold, floor(BUFFER\_DATA\_BUFFER\_SIZE / number of active axes)]}}{\text{25ms * frequency divider}}$$

The `LisAccelerometer` class is defined in src/isense/modules/security_module /lis_accelerometer.h in the iSense directory.

3.4.1. Constructor

```
LisAccelerometer::LisAccelerometer(Os &os);
```

Description:

Initializes the `LisAccelerometer` class. By default, the sensor is set into the continuous mode. All three axes are activated, the range is set to 2g, and the divider is set to 1, i.e. the sensor operates at a frequency of 40Hz, delivering a set of sensor readings for all axes every 25 milliseconds. The handler threshold, i.e. the maximum number of data sets in each buffer, is set to infinity, i.e. as many data sets as fit into the buffer are passed at each call to the set data handler. The sensor is not switched on until `enable` is called.

Parameters:

`os` Reference to the operating system class `Os`

Required modules:

Accelerometer* `#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

3.4.2. enable

```
bool LisAccelerometer::enable();
```

Description:

Switches on the accelerometer (regardless of prior state). Note that if no handler was set, the sensor is not enabled.

Parameters:

none

Returns:

`true` if the sensor was enabled, `false` otherwise (no data handler configured, or enable failed for another reason)

Required modules:

Accelerometer* `#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

3.4.3. disable

```
void LisAccelerometer::disable();
```

Description:

Switches off the accelerometer (regardless of prior state)

Parameters:

none

Required modules:

Accelerometer* `#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

3.4.4. set_mode

```
void LisAccelerometer::set_mode( uint8 mode);
```

Description:

Configures the sensor operation mode

Parameters:

mode	operation mode of the sensor	
	`MODE_CONTINUOUS`	continuous mode
	`MODE_THRESHOLD`	threshold mode for movement detection
	`MODE_FF`	freefall detection mode

Note:

If the sensor is in the movement of freefall detection mode, and an event is detected, the sensor changes into the continuous mode and starts filling a buffer. When the buffer is full, or the data set count reached the handler threshold, the `handle_buffer_data` of the set `BufferDataHandler` is called and the buffer is passed.

Required modules:

Accelerometer* `#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

3.4.5. set_threshold

```
void LisAccelerometer::set_threshold( uint16 threshold);
```

Description:

Configures the acceleration threshold (in mg) that must be exceeded to generate an event in the movement detection mode or undergone in the freefall detection more respectively. Note that threshold is not expressed as an absolute value for the movement detection, i.e. is a relative value describing the difference between two subsequent readings of one axis, while it is an absolute value for the free fall dectection.

Parameters:

`threshold` acceleration that, if exceeded/undergone, triggers the change to continuous mode

Required modules:

Accelerometer* `#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

3.4.6. set_handler_threshold

```
bool LisAccelerometer::set_handler_treshhold( uint8 handler_threshold );
```

Description:

Configures how many data sets the sensor stores in a buffer before it calls `handle_buffer_data` to pass the collected data sets to the data handler. Note that the new threshold is not set if `threshold` data sets would not fit into the buffer.

The maximum handler threshold can be calculated is follows:

max. threshold = floor(`BUFFER_DATA_BUFFER_SIZE` / number of active axes)

Parameters:

`handler_threshold`
number of data sets the sensor is to store in the buffer before calling the handler

Returns:

`true` if the new threshold was set, `false` otherwise

Required modules:

Accelerometer* `#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

3.4.7. set_divider

```
void LisAccelerometer::set_divider( uint8 divider );
```

Description:

Sets the sensor frequency divider.

Parameters:

`divider`	frequency divider. Setting 1 results in a frequency of 40Hz, setting 2 means the sensor reads every second value only (i.e. 20Hz) and so on

Required modules:

Accelerometer*	`#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

3.4.8. set_extended_range

```
void LisAccelerometer::set_extended_range( bool extended );
```

Description:

Sets the maximum range of the sensor.

Parameters:

`extended`	if `true`, the range is set to 6g, otherwise it is set to 2g

Required modules:

Accelerometer*	`#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

3.4.9. set_axes

```
void LisAccelerometer::set_axes(
                bool enable_x, bool enable_y, bool enable_z );
```

Description:

Sets the active axes.

Parameters:

`enable_x`	pass `true` to enable x-axis, false otherwise
`enable_y`	pass `true` to enable y-axis, false otherwise
`enable_z`	pass `true` to enable z-axis, false otherwise

Required modules:

Accelerometer*	`#define ISENSE_ENABLE_ACCELEROMTER`

Available for the targets:

JN5139R, JN5139R1

4. Security Module Demo Application

The Security Module demo application exemplifies how an application can make use of the `PirSensor` and the `LisAccelerometer` API. It shows how to

- configure the sensor,
- implement the event and data handlers
- send accelerometer data to iShell.

4.1. Obtaining the Security Module Demo Application

Go to coalesenses web site, click on "Downloads", and then on "iSense Hardware". In the "Security Module" section, click on the icon at the right in the "Security Module Demo Application" row, and download SecurityModuleDemoApplication.zip.

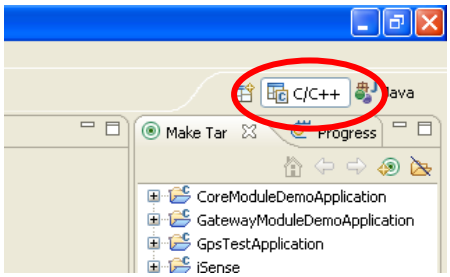
Extract the content of SecurityModuleDemoApplication.zip to the iApps directory.

4.2. Compiling the Security Module Demo Application

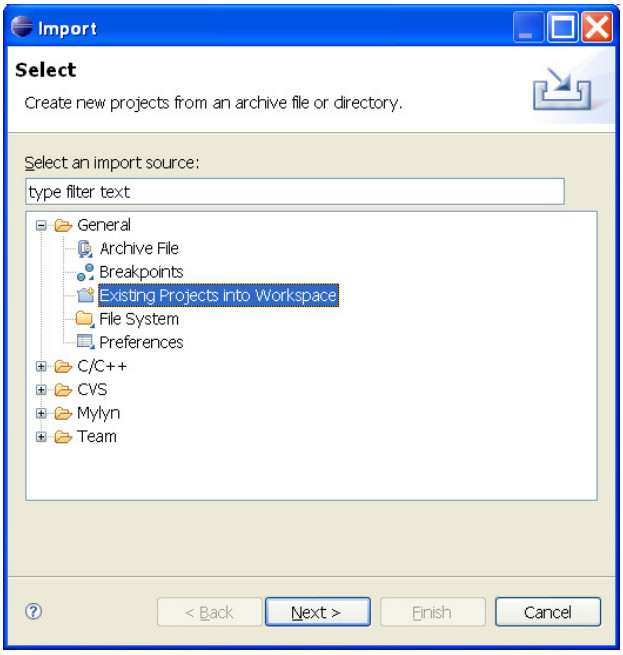
The next step is compiling the Security Module Demo Application.

4.2.1. Using Eclipse

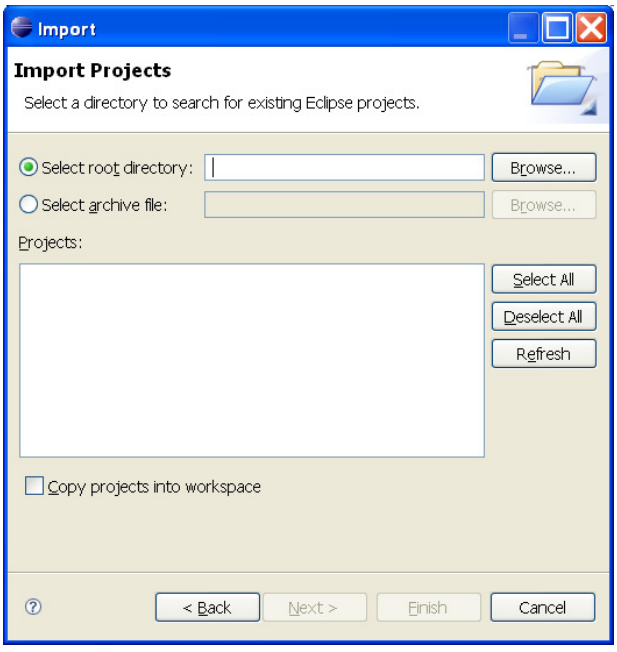
If you want to use Eclipse for compiling the Security Module Demo Application, open Eclipse, and change to the "C++" perspective.



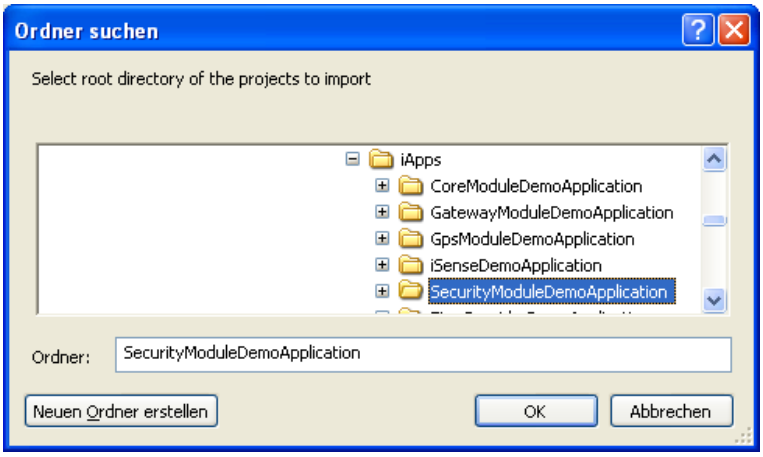
You can now import the application into Eclipse. Choose "File" → "Import" from the menu bar to open the "Import" dialog.



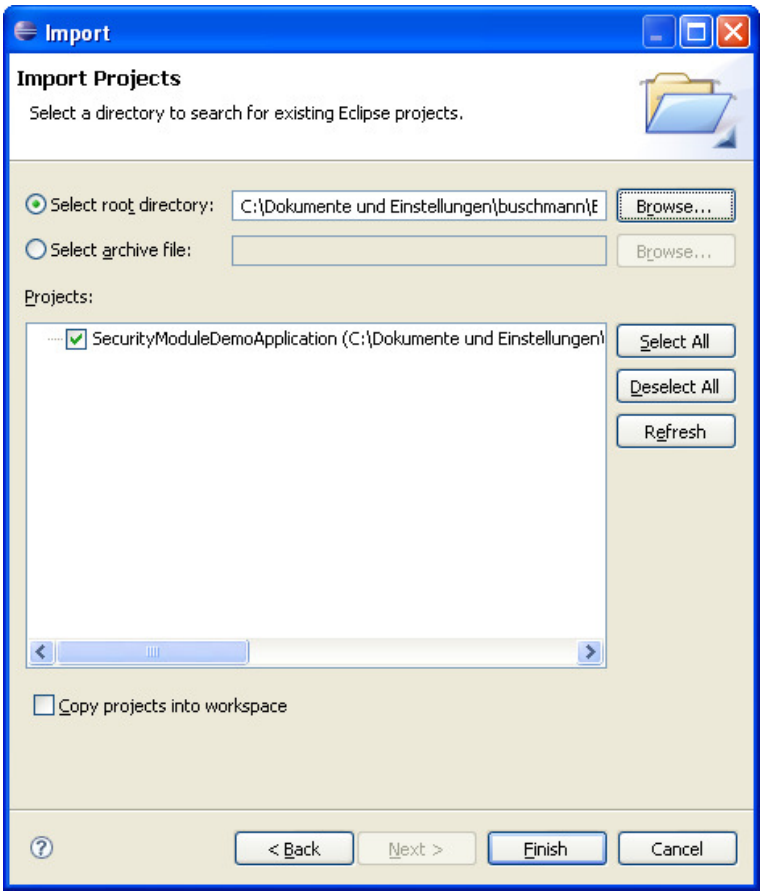
Choose “Existing projects into workspace”, and click on “Next”.



Click on the “Browse…”, and select the directory of the Security Module demo application, i.e. SecurityModuleDemoApplication in the iApps directory.



After doing do, the "SecurityModuleDemoApplication" project should appear in the projects list of the "Import" dialog.

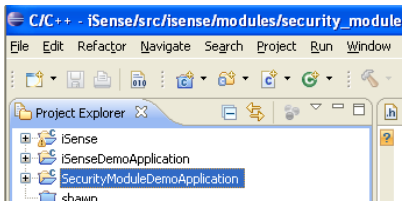


If it doesn't, the typical reason is that there is already a project called "SecurityModuleDemoApplication" in Eclipse, i.e.

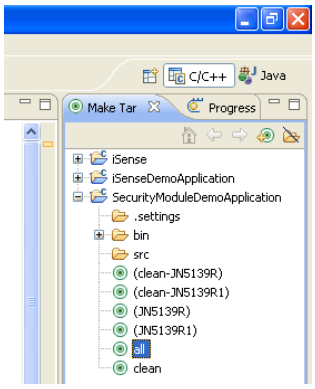
- you imported the SecurityModuleDemoApplication before or
- you created or imported a project with the same name before.

Be sure not to check “Copy projects into workspace” in the above dialog. Finish the import by clicking on the “Finish” button.

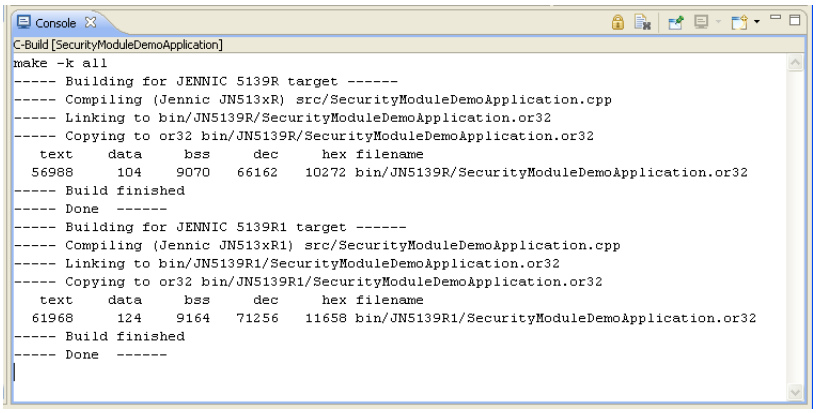
As a result, you will find the “SecurityModuleDemoApplication” project in the “Project Explorer”.



In addition, the “SecurityModuleDemoApplication” will appear in the “Make Targets” view. Double click on “all” to build the demo application for all targets.

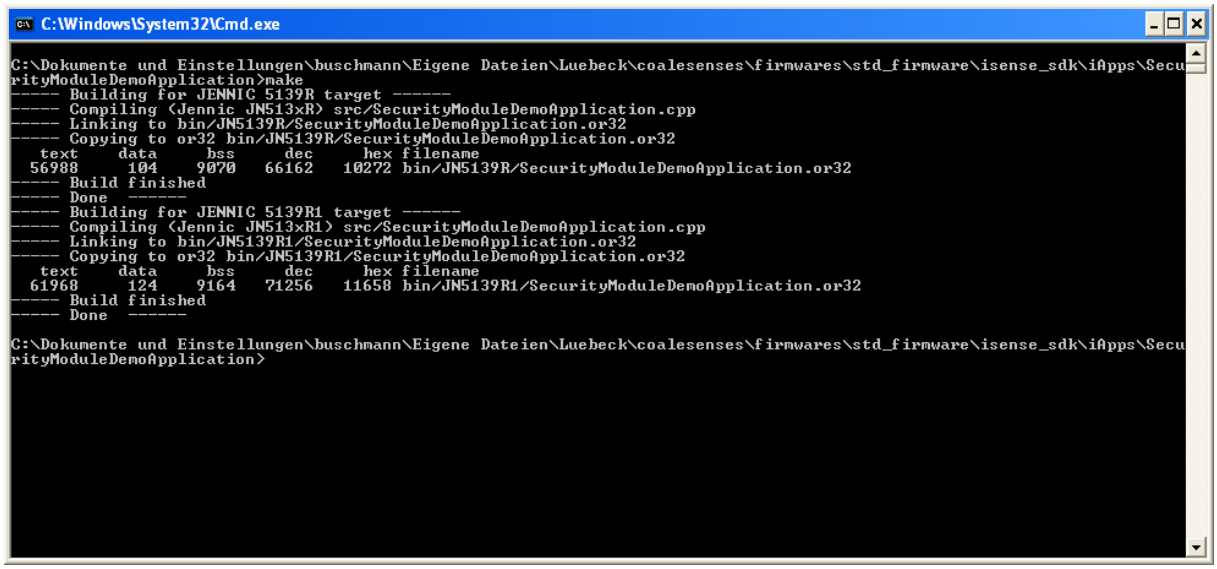


As a result, the “Console” view displays the compiler output during the build process. After that, it should look similar to the output depicted below.


Console
C-Build [SecurityModuleDemoApplication]
make -k all
----- Building for JENNIC 5139R target ------
----- Compiling (Jennic JN513xR) src/SecurityModuleDemoApplication.cpp
----- Linking to bin/JN5139R/SecurityModuleDemoApplication.or32
----- Copying to or32 bin/JN5139R/SecurityModuleDemoApplication.or32
text data bss dec hex filename
56988 104 9070 66162 10272 bin/JN5139R/SecurityModuleDemoApplication.or32
----- Build finished
----- Done ------
----- Building for JENNIC 5139R1 target ------
----- Compiling (Jennic JN513xR1) src/SecurityModuleDemoApplication.cpp
----- Linking to bin/JN5139R1/SecurityModuleDemoApplication.or32
----- Copying to or32 bin/JN5139R1/SecurityModuleDemoApplication.or32
text data bss dec hex filename
61968 124 9164 71256 11658 bin/JN5139R1/SecurityModuleDemoApplication.or32
----- Build finished
----- Done ------

4.2.2. Using the Command Line

If you want to use the command line tools for building the Security Module Demo Application, open a console window, change to the SecurityModuleDemoApplication directory in your iApps directory, and type “make”.


C:\Windows\System32\Cmd.exe
C:\Dokumente und Einstellungen\buschmann\Eigene Dateien\Luebeck\coalesenses\firmwares\std_firmware\isense_sdk\iApps\SecurityModuleDemoApplication>make
----- Building for JENNIC 5139R target ------
----- Compiling (Jennic JN513xR) src/SecurityModuleDemoApplication.cpp
----- Linking to bin/JN5139R/SecurityModuleDemoApplication.or32
----- Copying to or32 bin/JN5139R/SecurityModuleDemoApplication.or32
text data bss dec hex filename
56988 104 9070 66162 10272 bin/JN5139R/SecurityModuleDemoApplication.or32
----- Build finished
----- Done ------
----- Building for JENNIC 5139R1 target ------
----- Compiling (Jennic JN513xR1) src/SecurityModuleDemoApplication.cpp
----- Linking to bin/JN5139R1/SecurityModuleDemoApplication.or32
----- Copying to or32 bin/JN5139R1/SecurityModuleDemoApplication.or32
text data bss dec hex filename
61968 124 9164 71256 11658 bin/JN5139R1/SecurityModuleDemoApplication.or32
----- Build finished
----- Done ------
C:\Dokumente und Einstellungen\buschmann\Eigene Dateien\Luebeck\coalesenses\firmwares\std_firmware\isense_sdk\iApps\SecurityModuleDemoApplication>

After the build finished, the console output should look similar as shown above.

4.3. Flashing the Security Module Demo Application

Connect either

- an iSense Security Module with or without a PIR sensor
- an iSense Core Module and
- an iSense Gateway Module with a USB cable

or

- an iSense Security Module with or without a PIR sensor
- an iSense Core Module,
- an iSense Battery Module and
- an iSense Gateway Module with a serial cable

and plug the USB/serial cable to the PC. Be sure that the Core Module is switched on.

Start iShell, configure the correct communication port, and change to the “Flash Loader” view. Select the appropriate binary according to the table below.

Chip	*Binary file*
JN5139R	iApps/SecurityModuleDemoApplication/bin/JN5139R/SecurityModuleDemoApplication.bin
JN5139R1	iApps/SecurityModuleDemoApplication/bin/JN5139R1/SecurityModuleDemoApplication.bin

For details regarding the identification of the chip version, refer to Section 2.4 of the *iSense Core Module User Guide* [4]. Then flash the binary to the connected iSense sensor node.

4.4. Security Module Demo Application functionality

The source file of the Security Module demo application is located at SecurityModuleDemoApplication/src/SecurityModuleDemoApplication.cpp. The class `SecurityModuleApplication` is derived from `isense::Application`, and additionally inherits from `isense::BufferDataHandler` and `isense::SensorHandler`.

```
class SecurityModuleDemoApplication :
      public Application,
      public BufferDataHandler,
      public SensorHandler
{
      public:
            // constructor
            SecurityModuleDemoApplication(Os &os);

            // inherited from application, called upon device boot
            void boot();

            // inherited from BufferDataHandler, called when
            // accelerometer data is available
            virtual void handle_buffer_data( BufferData* buf_data );

            // inherited from SensorHandler, called when a
            // passive infrared sensor event occurs
            virtual void handle_sensor();
      private:
            // pointer to the accelerometer
            LisAccelerometer* accelerometer_;
            // pointer to the passive infrared (PIR) sensor
            PirSensor* pir_;
            // pointer to the iShell interpreter
            IShellInterpreter* isi_;
};
```

`SecurityModuleDemoApplication` has three member variables that are initialized in the constructor. `accelerometer_` holds the pointer to the `LisAccelerometer` instance, `pir_` points to a `PirSensor` instance, and `isi_` to an IShellInterpreter instance.

```
SecurityModuleDemoApplication::
SecurityModuleDemoApplication(Os &_os) :
      Application(_os),
      accelerometer_(NULL),
      pir_(NULL),
      isi_(NULL)
{
}
```

`SecurityModuleDemoApplication` overwrites `isense::Application::boot()`, which is called upon the start up of the iSense sensor node.

```
void
SecurityModuleDemoApplication::
boot ()
{
      // output boot notification message
      os().debug("Booting Security Module Demo Application, id=%x",
                                                              os().id());

      // create LisAccelerometer instance
      accelerometer_ = new LisAccelerometer(os());
      // create PriSensor2 instance
      pir_ = new PirSensor(os());
      // create iShell interpreter instance
      // (needed to send accelerometer readings to iShell)
      isi_ = new IShellInterpreter(os());

      if ((accelerometer_ != NULL) && (pir_!= NULL) && (isi_!=NULL))
      {
            // ----- configure accelerometer ------
            // set the threshold to 5 mg
            accelerometer_->set_threshold(5);

            // set threshold mode, i.e. the sensor will start to
            // deliver data after set threshold was exceeded
            accelerometer_->set_mode(MODE_THRESHOLD);

            // set this application as the sensor event handler
            // --> handle_buffer_data will be called if
            // sensor data is available
            accelerometer_->set_handler(this);

            // switch accelerometer
            accelerometer_->enable();

            // ----- configure PIR sensor -------------
            // set this application as the sensor event handler
            // --> handle_sensor will be called upon a PIR event
            pir_->set_sensor_handler(this);
            //set the PIR event duration to 2 secs
            pir_->set_pir_sensor_int_interval( 2000 );
            // switch on the PIR sensor
            pir_->enable();
      } else
            os().fatal("Could not allocate PIR/accelerometer/iShell
interpreter");
}
```

Within the `boot()` method, at first a boot notification message is output.

You can send character output via the Gateway Module or the radio to the outside world using the methods `Os::debug` and `Os::fatal`:

```
//---------------------------------------------------------
** Logs the given string depending on the log mode to a uart or radio.
 */
```

```
void debug( const char *format, ...);

//-------------------------------------------------------
/** Logs the given string depending on the log mode to a uart or radio.
 */
void fatal( const char *format, ...);
```

They work similar to the well-known `sprintf` functions in regular C, i.e. integer values etc. can be printed using the % notation. For example

```
uint16 i = 128;
os().debug("value of i is %d, hex=%x", i, i);
```

will output „value of i is 128, hex=0x80“. The destination of the debug output can be set using `Os::set_log_mode`, the corresponding constants can be found in iSense/src/isense/os.h.

```
//-------------------------------------------------------
/** Sets the log mode to the given value, e.g.
 * (ISENSE_LOG_MODE_UART0 | ISENSE_LOG_MODE_RADIO)
 */
void set_log_mode( uint8 mode ) {log_mode_ = mode; }
```

By default, the output destination is set to UART0, i.e. the output is sent to a PC via a connected iSense Gateway Module.

Then, in `SecurityModuleDemoApplication::boot()`, the three objects that represent the accelerometer, the PIR sensor as well as the `IShellInterpreter` are allocated. The class `IShellInterpreter` provides a suite of methods that can be used to conveniently send or receive data to/from iShell.

If all three allocations were successful, the sensors will be configured, otherwise an fatal error message is output.

```
void
SecurityModuleDemoApplication::
boot ()
{

      :

      // create LisAccelerometer instance
      accelerometer_ = new LisAccelerometer(os());
      // create PriSensor2 instance
      pir_ = new PirSensor(os());
      // create iShell interpreter instance
      // (needed to send accelerometer readings to iShell)
      isi_ = new IShellInterpreter(os());
```

```
      if ((accelerometer_ != NULL) && (pir_!= NULL) && (isi_!=NULL))
      {
            // ----- configure sensors ------

            :
      } else
            os().fatal("Could not allocate sensors/iShell interpreter");
}
```

Within the if-clause, the accelerometer is configured. At first, sensor is set to the threshold mode for movement detection. When the change in subsequent sensor readings exceeds a threshold, the sensor will start reading out values and pass them to the data handler. However, static absolute accelerations (such as induced by earth's gravitational field) will be ignored, so that the sensor is exclusively sensitive to movement or vibration.

After that, the threshold for the motion detection is set to 5 mg. This will cause an event to occur event at the slightest change of sensor values, i.e. in case of tiny movements. For generating events e.g. on rough shock, the threshold could be set to 500 or 1000 mg. When an event is generated, the sensor will automatically change into the continuous mode, and will start collecting sets of sensor readings for all three axes.

When the buffer is full, `SecurityModuleDemoApplication::handle_buffer_data` will be called because the application is set as the sensors data handler. Finally, the accelerometer is enabled.

Then, the PIR sensor is configured. At first, the application is set as the sensor's event handler, resulting in `SecurityModuleDemoApplication::handle_sensor` being called upon sensor events. The interrupt suppression interval is set to two seconds, and the sensor is enabled (it will to ready for surveillance after a stabilization time of four seconds).

```
            // ----- configure accelerometer ------
            // set threshold mode, i.e. the sensor will start to
            // deliver data after set threshold was exceeded
            accelerometer_->set_mode(MODE_THRESHOLD);

            // set the threshold to 5 mg
            accelerometer_->set_threshold(5);

            // set this application as the sensor event handler
            // --> handle_buffer_data will be called if
            // sensor data is available
            accelerometer_->set_handler(this);

            // switch accelerometer
            accelerometer_->enable();

            // ----- configure PIR sensor -------------
            // set this application as the sensor event handler
            // --> handle_sensor will be called upon a PIR event
            pir_->set_sensor_handler(this);
            //set the PIR event duration to 2 secs
            pir_->set_pir_sensor_int_interval( 2000 );
            // switch on the PIR sensor
            pir_->enable();
```

When the accelerometer changed to the continuous mode (induced by a threshold exceedance) and filled the first buffer, `SecurityModuleDemoApplication::handle_buffer_data` is called and the collected data is passed.

```
void
SecurityModuleDemoApplication::
handle_buffer_data( BufferData* buf_data )
{
      os().debug("ACC");
      // send data to iShell. Sampling frequency
      // is 40 Hz --> inter-reading spacing is 25 ms
      isi_->send_buffer_data(buf_data, os().id(), 25);

      //return from continuous mode to threshold mode
      accelerometer_->set_mode(MODE_THRESHOLD);
}
```

A debug message (“ACC”) is output, and then the data is sent to iShell.

```
#ifdef ISENSE_ENABLE_ISI_BUFFERSENDING
/**
 * Send the buffer_data to iShell with packet type
 * isense::Uart::MESSAGE_TYPE_INT_BUFFER
 *
 * \param buffer_data The buffer with the data to be sent to iShell.
 * \param id The id of the corresponding device.
 * \param interval The time in milliseconds between two values.
 */
bool send_buffer_data( BufferData* buffer_data,
                       uint16 id,
                       uint16 interval);

#endif
```

`IShellInterpreter::send_buffer_data` takes the actual `buf_data`, the ID of the sending device (returned by `Os::id()`, to display it in iShell) as well as the inter-value spacing as parameters. It then sends the data to UART0 (similar to `Os::debug`), so if an iSense Gateway Module is connected to the node, the data can be forwarded to iShell. If the “Curve illustrator” plugin is active, it will receive the data and display the according graphs for all axes that are active at the accelerometer.

In `SecurityModuleDemoApplication::handle_buffer_data` the accelerometer finally is set back into the threshold mode. Like this, no further sensor data will be delivered until the next threshold exceedance. However, if the sensor node is moved continuously, the threshold will be exceeded again immediately, the sensor will again start collecting sensor data sets, and send them to iShell, so that the charts in the “Curve illustrator” will continuously display the sensor data.

Otherwise, the curve illustrator will leave a gap between the different sequences of sensor readings.

If a PIR sensor event occurs, `SecurityModuleDemoApplication::handle_sensor` is called. The demo application just outputs a short message here.

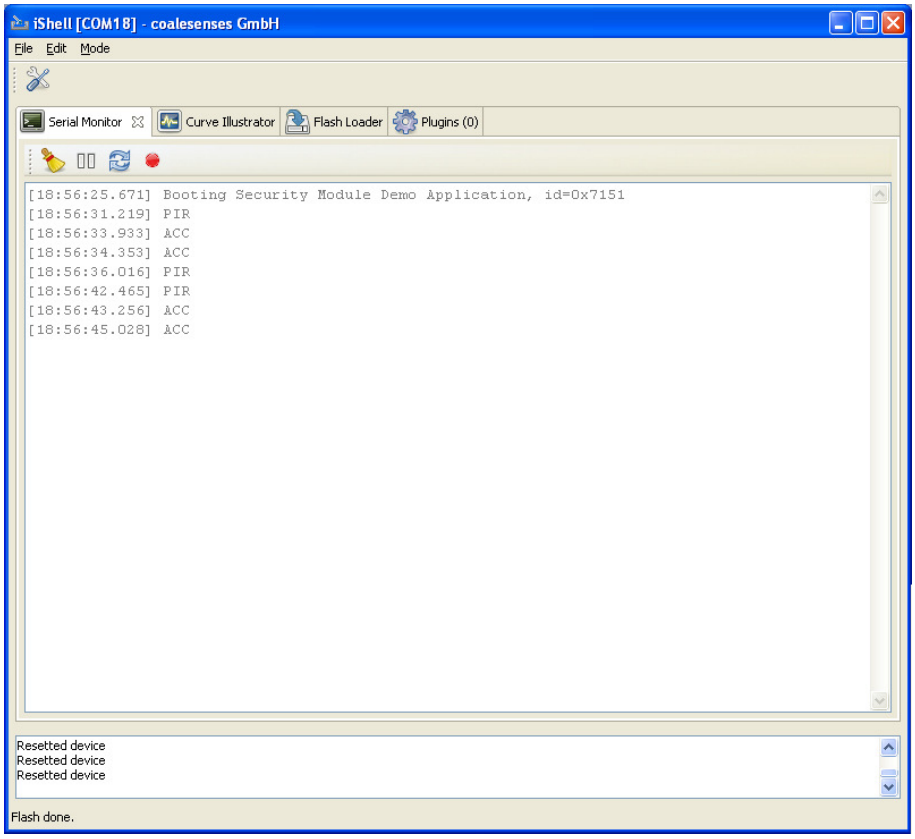
```
void
SecurityModuleDemoApplication::
handle_sensor()
{
      os().debug("PIR ");
}
```

The `SecurityModuleDemoApplication` is instantiated in the `application_factory` method. It is called by the iSense operating and networking firmware when the devices boots, and must be implemented for every application. In other words, the `application_factory` method is the way to let the firmware know which application to instanciate.

```
Application* application_factory(Os &os)
{
      //create application instance
      return new SecurityModuleDemoApplication(os);
}
```

All in all, the `SecurityModuleDemoApplication` configures the PIR sensor and the accelerometer to issue alarms upon temperature fluctuations and node movements.

When the PIR sensor generates an event, the application outputs a “PIR” message to the “Serial Monitor” plugin of iShell. Upon node movement, the application outputs “ACC”.



In addition, the application sends the 16 sensor readings for all three axes following a movement detection event to the “Curve illustrator” plugin of iShell (if it is activated).

For additional information on iShell and its plugins, please refer to the *iShell User Guide* [2].

5. References

[1] coalesenses Development Environment Setup User Guide, online available at http://www.coalesenses.com/download/UG_development_environment_setup_v1.9_web.pdf

[2] coalesenses iShell User Guide, online available at http://www.coalesenses.com/download/UG_ishell_v1.3.pdf

[3] coalesenses Writing iSense Applications User Guide, online available at http://www.coalesenses.com/index.php?page=isense-applications

[4] coalesenses iSense Core Module User Guide, online available at http://www.coalesenses.com/index.php?page=hardware-docs-and-demos

coalesenses GmbH
Maria-Goeppert-Str. 1
23562 Lübeck
Germany

www.coalesenses.com
sales@coalesenses.com